# Version Control Systems

Система управления версиями (Version Control System, VCS) — ПО для облегчения работы с изменяющейся информацией. VCS позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

## Типы VCS

* **Local** (RCS)

- хранит различия между файлами (патчи) в локальной базе данных

* **Centralized** (CVS, Subversion (SVN), Perforce)

- один сервер и клиенты, которые с него берут файлы.

Плюсы: всегда понятно кто что делает в текущий момент

Минусы: сервер - слабое место, при его поломке практически все данные будут утеряны (кроме небольших кусочков), если он не работает или нет интернета, то сохранить ничего нельзя.

* **Distributed** (Git, Mercurial, Bazaar, Darcs)

- клиент является полным зеркалом сервера, а не просит у него по одному файлу. Как следствие он может полностью работать локально и заливать на сервер свои изменения, только когда есть такая возможность.

## Git

Главное отличие Git от других систем контроля версий - это снимки состояний (snapshot) вместо отличий (differences). Если файл не изменялся, то просто хранится ссылка на его предыдущую версию.

Для всех файлов и своих внутренних объектов гит использует контрольную сумму - посчитанный с помощью SHA-1 хеш файла (40 16-ричных символов).

Но так как в рамках одного репозитория часто достаточно первых 4-5 цифр, то можно попросить git укоротить хеш не потеряв его уникальность.

git rev-parse --short=4 <hash>

## Simple file workflow

Статус каждого файла в проекте всегда либо untracked, либо tracked.

Любой tracked файл может находиться хотя бы в одном из следующих состояний:

* Modified - просто измененный файл
* Staged - файл, помеченный для последующего коммита. Такой файл попадает в staging area (или другое название – index)
* Commited - файл, сохраненный в локальной базе данных гита

Modified 
Unt racked 
unmodif i ed 
Ed It the file 
FIGURE 2-1 
St aged 
The lifewcle of the 
Status of your files. 
the tile 

git **init** - создание репозитория в папке

git **add** <files\_pattern> - помещение файла в staging area

git **status** - проверка состояния

git **commit** - отправка изменений в репозиторий

**--amend** позволяет отредактировать предыдущий коммит (важно это делать только для коммитов, которых еще нет в удаленном репозитории), а именно добавит в него все изменения из staged area и даст поменять сообщение.

git **log** - просмотр списка коммитов

У команды log довольно много флагов, которые помогут улучшить ее вывод. Например, можно использовать git log --oneline --decorate --graph --all

**git <command> --help**

## Команды для отмены изменений

git **checkout** <hash or branch>

откат к коммиту <hash> или последнему коммиту ветки <branch>.

**Notes:**

* После checkout-а на коммит вы попадете в состояние, называемое Detached HEAD, в котором нельзя добавлять файлы в stage area и, следовательно, коммитить.
* После checkout все файлы перезаписываются к нужному состоянию, поэтому в большинстве случаев невозможно выполнить checkout при наличии несохраненных изменений.

git **reset** <hash>

--soft

перемещает текущую ветку на нужный коммит. В директории ничего не меняется. Все откатившиеся файлы попадают в staging area.

--mixed (default)

делает то же самое, что и --soft, а после этого unstage все staged файлы

--hard (**опасно!!!**)

полностью откатывает все файлы к указанному коммиту.

git **revert** <hash>

создает коммит полностью противоположный указанному

git **rm** <file>

удаляет файл и добавляет это изменение в staged area (вместо файла можно передать паттерн, но \* в нем нужно экранировать)

--cached – удаляет файл только из индекса (переводит в untracked)

--force – удаляет файл даже если в нем есть изменения (просто rm удалять такой файл откажется)

git **mv** <source> <dest>

перемещает/переименовывает файлы и добавляет в staged area

Если переименовать файл руками, то git поймет это как удаление файла со старым именем, а файл с новым именем будет unstaged. Поэтому придется делать git add для нового файла и git rm –cached для старого.

## Работа с ветками

git **branch** <name> - создание новой ветки

git **checkout** <branch> - перемещение указателя на ветку

**-b** создаст ветку и переместит на нее указатель

git **merge** <feature\_branch> - слияние двух веток

git **rebase** <base\_branch>

берет все отличающиеся коммиты из текущей ветки и применяет их после коммитов из base\_branch.

Коммиты, получившиеся в результате rebase будут такими же, но иметь другой hash. Как следствие если они уже были в удаленном репозитории, то при push git попытается смержить старые и новые коммиты. Чтобы он этого не делал нужно использовать push с флагом –force, но это опасно, потому что просто затрет удаленную ветку, что может повредить чужие коммиты. Поэтому rebase рекомендуется делать только на ветках, которых еще нет в удаленном репозитории.

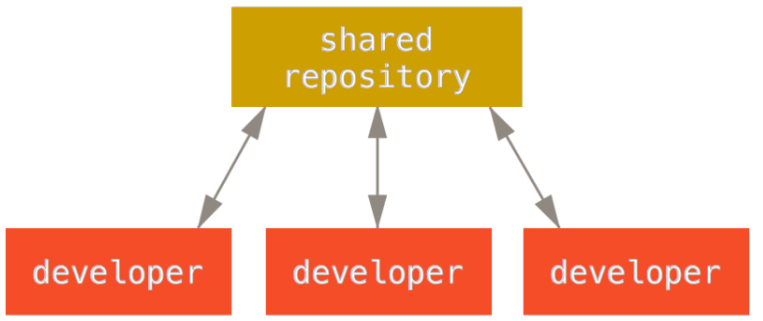
git **cherry-pick** <hash>

делает patch из коммита и пытается его применить на текущую ветку.

git **stash**

скрывает все сделанные изменения во временное хранилище и переводит код в состояние HEAD. Применить скрытое изменение можно с помощью apply или pop. См. git stash --help.

## Удаленные репозитории



git **clone** - клонирование существующей репозитории

git remote **add** <remote\_name> <url>

добавление репозитории к существующему проекту (обычно в качестве <remote\_name> используют origin и при clone оно по умолчанию будет именно таким)

git **push** – отправка изменений на сервер

Если текущая ветка еще не существует на сервере, то нужны параметры <remote> и <branch>, например origin feature1. Если дополнительно добавить флаг –u (--set-upstream), то локальная ветка и remote ветка свяжутся и больше при push ее указывать не придется.

git **fetch** – скачивание изменений с сервера

git **pull** – fetch + merge

## .gitignore

A **.gitignore** file specifies intentionally untracked files that Git should ignore.

**Rules for .gitignore file:**

• Blank lines or lines starting with # are ignored.

• Standard glob patterns work.

• You can start patterns with a forward slash (/) to avoid recursivity.

• You can end patterns with a forward slash (/) to specify a directory.

• You can negate a pattern by starting it with an exclamation point (!).

**Glob patterns:**

• An asterisk (\*) matches zero or more characters.

• [abc] matches any character inside the brackets (in this case a, b, or c);

• a question mark (?) matches a single character.

• and brackets enclosing characters separated by a hyphen ([0-9]) matches any character between them.

• use two asterisks (\*\*) to match nested directories.

## Дополнительные возможности

## Git config

3 уровня конфигурационных файлов (по возрастанию приоритета):

1. --system (C:/ProgramData/Git/.config)

2. --global (C:/Users/$USER/.gitconfig)

3. config file in the Git directory (.git/config)

**e.g.**

git config –global user.name "John Doe"

git config –global user.email johndoe@example.com

core.ignorecase false (по умолчанию true)

alias.<aliasName> <command> (позволяет задать псевдоним для вызова <command>. При этом command может быть комплексной, т.е. с флагами и прочим)

git config –list (список настроек)

git config --edit (позволяет напрямую просмотреть или отредактировать содержимое файла конфигурации)

## Git hooks

client side (pre-commit, post-commit, pre-push, etc.)

server side (pre-receive, update, post-receive)